PR #46862 完整报告

vllm-project/vllm

[GLM5.2 Perf] `fused_indexer_q_rope_quant` triton kernel, 1.9% ~ 3.3% E2E Throughput improvement.

合并时间：2026-06-27 13:16

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/46862

执行摘要

- 一句话：融合 Q RoPE、FP8 量化与权重缩放 Triton kernel，提升 GLM-5.2 吞吐 1.9%-3.3%
- 推荐动作：值得精读该 Triton kernel 的融合实现，尤其是在 attention 量化场景下的编程技巧。对于未来支持其他模型（如 DeepSeek 系列）的类似优化有直接参考价值。

功能与动机

PR body 指出 GLM-5.2 模型有与 DeepSeek V4（DSv4）类似的融合 kernel 需求。原来流程为：Q RoPE → cat → FP8 quant → q_scale fold into weights，现在在单个 Triton kernel 内完成所有步骤，减少 launch 开销和显存带宽。

实现拆解

1. 在 vllm/model_executor/layers/sparse_attn_indexer.py 中新增 Triton JIT kernel _fused_indexer_q_rope_quant_kernel 和 Python wrapper 函数 fused_indexer_q_rope_quant。Kernel 内部按 token 和 head 并行，首先根据布局（NeoX 或 interleaved）加载 Q 的 rope 部分，应用 RoPE 旋转，然后与 nope 部分合并计算 per-token 的 FP8 量化 scale（e8m0 格式，向上取整到 2 的幂），统一量化写入 fp8 输出 buffer；同时将 q_scale 折叠到 weights 中（乘以 softmax_scale 和 head_scale）。
2. 在 deepseek_v2.py 的 DeepseekV2Attention 类中添加 use_fused_indexer_q 开关，生效条件为：CUDA 平台、quant_block_size == head_dim == 128、rope_dim == 64、scale_fmt 非空。在 forward 方法中新增 elif 分支，先通过一次 GEMM 获得 k 和 weights，然后调用融合 kernel 得到量化后的 q_fp8 和带 scale 的 weights，再对 k 的 rope 部分单独旋转后与 nope 部分拼接，最后调用 self.indexer_op 完成剩余操作。
3. 精度和性能验证：使用 GLM-5.2-FP8 模型，通过 lm_eval gsm8k 任务验证精度（0.9439 exact_match），通过 vllm bench 对比 main 分支，显示吞吐提升 1.9%-3.3%，TTFT 降低约 5%。
4. 测试配套：未新增独立单元测试，但通过集成测试和 benchmark 验证。

关键文件：

- vllm/model_executor/layers/sparse_attn_indexer.py（模块 注意力层；类别 source；类型 core-logic；符号 _fused_indexer_q_rope_quant_kernel,

fused_indexer_q_rope_quant）：核心变更，新增 Triton JIT kernel，实现 RoPE、FP8 量化、权重缩放融合

- vllm/model_executor/models/deepseek_v2.py（模块 DeepSeek 模型；类别 source；类型 core-logic）：修改模型 forward 分支，调用融合 kernel，添加条件开关

关键符号：_fused_indexer_q_rope_quant_kernel, fused_indexer_q_rope_quant

关键源码片段

vllm/model_executor/layers/sparse_attn_indexer.py

核心变更，新增 Triton JIT kernel，实现 RoPE、FP8 量化、权重缩放融合

融合 kernel：_fused_indexer_q_rope_quant_kernel

```
@triton.jit
def _fused_indexer_q_rope_quant_kernel(
    positions,
    q,
    q_s0,
    q_s1,
    cos_sin_cache,
    cos_sin_s0,
    q_fp8,
    q_fp8_s0,
    q_fp8_s1,
    weights,
    weights_s0,
    weights_s1,
    weights_out,
    weights_out_s0,
    weights_out_s1,
    softmax_scale,
    head_scale,
    fp8_min: tl.constexpr,
    fp8_max: tl.constexpr,
    is_neox: tl.constexpr,
):
    token = tl.program_id(0)
    head = tl.program_id(1)
    offs32 = tl.arange(0, 32)
    offs64 = tl.arange(0, 64)

    pos = tl.load(positions + token)
    cos = tl.load(cos_sin_cache + pos * cos_sin_s0 + offs32).to(tl.float32)
    sin = tl.load(cos_sin_cache + pos * cos_sin_s0 + 32 + offs32).to(tl.float32)
    q_base = q + token * q_s0 + head * q_s1
    out_base = q_fp8 + token * q_fp8_s0 + head * q_fp8_s1

    if is_neox:
```

```
    # NeoX 布局：前半部分 0-31 为 x0，后半部分 32-63 为 x1
    x0 = tl.load(q_base + offs32).to(tl.float32)
    x1 = tl.load(q_base + 32 + offs32).to(tl.float32)
else:
    # 交错布局：x0 取偶数索引，x1 取奇数索引
    x0 = tl.load(q_base + offs32 * 2).to(tl.float32)
    x1 = tl.load(q_base + offs32 * 2 + 1).to(tl.float32)

# 应用 RoPE 旋转，中间用 bfloat16 降低精度损耗
r0 = (x0 * cos - x1 * sin).to(tl.bfloat16).to(tl.float32)
r1 = (x1 * cos + x0 * sin).to(tl.bfloat16).to(tl.float32)
amax = tl.maximum(tl.max(tl.abs(r0)), tl.max(tl.abs(r1)))

# 处理 nope 部分（不进行旋转）
q_nope = tl.load(q_base + 64 + offs64).to(tl.float32)
amax = tl.maximum(amax, tl.max(tl.abs(q_nope)))

# 计算量化 scale（e8m0 格式：向上取整到 2 的幂）
scale_raw = tl.maximum(amax, 1e-10) * (1.0 / fp8_max)
q_scale = tl.math.exp2(tl.ceil(tl.log2(scale_raw)))

# 存储量化后的 Q（包含 rope 和 nope 部分）
if is_neox:
    tl.store(out_base + offs32, tl.clamp(r0 / q_scale, fp8_min, fp8_max).to(q_fp8.dtype.
    element_ty))
    tl.store(out_base + 32 + offs32, tl.clamp(r1 / q_scale, fp8_min, fp8_max).to(q_fp8.dtype.
    element_ty))
else:
    tl.store(out_base + offs32 * 2, tl.clamp(r0 / q_scale, fp8_min, fp8_max).to(q_fp8.dtype.
    element_ty))
    tl.store(out_base + offs32 * 2 + 1, tl.clamp(r1 / q_scale, fp8_min, fp8_max).to(q_fp8.
    dtype.element_ty))
tl.store(out_base + 64 + offs64, tl.clamp(q_nope / q_scale, fp8_min, fp8_max).to(q_fp8.dtype.
element_ty))

# 将 q_scale 折叠到权重中：weight * q_scale * softmax_scale * head_scale
weight = tl.load(weights + token * weights_s0 + head * weights_s1).to(tl.float32)
tl.store(weights_out + token * weights_out_s0 + head * weights_out_s1,
         weight * q_scale * softmax_scale * head_scale)
```

模型 forward 中的调用分支

```
    elif self.use_fused_indexer_q and q.dtype == torch.bfloat16:
        # 融合 wk + weights_proj：一次 GEMM，然后拆分
        kw, _ = self.wk_weights_proj(hidden_states)
        k = kw[:, :self.head_dim]
        weights = kw[:, self.head_dim:]

        k = self.k_norm(k)
        k_pe, k_nope = torch.split(
```

```
        k, [self.rope_dim, self.head_dim - self.rope_dim], dim=-1
    )

    # 调用融合 kernel：一次完成 Q 的 RoPE、FP8 量化和 scale 折叠
    q_fp8, weights = fused_indexer_q_rope_quant(
        positions,
        q,
        rotary_emb.cos_sin_cache,
        weights,
        self.softmax_scale,
        self.n_head ** -0.5,
        rotary_emb.is_neox_style,
    )

    # 对 K 的 rope 部分单独旋转（MQA 风格，unsqueeze 后旋转再 squeeze）
    q_dummy = torch.empty_like(k_pe.unsqueeze(1))
    _, k_pe = rotary_emb(positions, q_dummy, k_pe.unsqueeze(1))
    k_pe = k_pe.reshape(-1, 1, self.rope_dim)
    k = torch.cat([k_pe.squeeze(-2), k_nope], dim=-1)

    return self.indexer_op(hidden_states, q_fp8, k, weights)
```

评论区精华

Review 中 tlrmchlsmth 在 sparse_attn_indexer.py 第 93 行评论："Should this be 1e-4 to match DSv4's?"，作者 yewentao256 回复："We here should use 1e-10 to match the current eps with per token group quant fp8... So no behavior change"。最终确认 eps 使用 1e-10 与既有量化函数一致。

- eps value alignment (correctness): 采用 1e-10，与既有量化函数一致

风险与影响

- 风险：
  1. 新 kernel 仅在特定条件启用（CUDA、head_dim=128 等），其他情况回退到原路径，不会引入功能退化。
  2. 缺少单元测试，但通过精度 (bench) 验证，且融合逻辑与原路径数值差异极小 (eps 一致)。
  3. 未来模型配置变化（如 head_dim 非 128）会导致自动回退，需确保回退路径表现正常。
  4. kernel 使用 Triton，对 ROCm 平台不生效，但已有 ROCm 专用路径。 - 影响：直接使用 GLM-5.2-FP8 模型的用户可获得 1.9%-3.3% 吞吐提升，TTFT 略降，无功能变化。代码量小，维护成本可控。该融合 kernel 为类似场景的优化提供参考模式。 - 风险标记：特定形状依赖，缺少单元测试

关联脉络

- PR #46808 [GLM-5] Add DSV3.2/GLM5 to vllm/models/: 为 GLM 模型添加了基础模型实现，本 PR 在该模型上进一步做性能优化